1. In the below elements which of them are values or an expression?

(Hint :- values can be integer or string and expressions will be mathematical operators).

a) * b)'hello' c)-87.8 d) - e) / f) + g) 6

Ans.

In python programming language, values can be integer, float or string and expressions will be mathematical operators.

Hence, b) 'hello' (string) c) -87.8 (float) g) 6 (integer) are values.

While, a)* (multiplication) d)- (subtraction) e)/ (Division) f)+ (Addition) are expressions.

2. What is the difference between string and variable?

Ans.

String:

A string in Python is a sequence of characters enclosed within either single (') or double ("") or triple (''''''') quotation marks. Strings can contain letters, numbers, symbols, and spaces, and they are used to represent textual data.

Variable:

A variable in Python is a named identifier that can hold a value. The value stored in a variable can be of various types, including strings, numbers, lists, dictionaries, etc. Variables are used to store and manipulate data within a program.

Eg.

In [1]:
```
s = 'Hello' # here s is a varible that stores string data type while Hello enclose

print(s)
```

```
Hello
```

3. Describe three different data types.

Ans.

Integer (int):

An integer is a whole number, both positive and negative, without any decimal point. Integers are used to represent quantities that don't have fractional parts.

Float:

A float (floating-point number) is a number with a decimal point. It is used to represent real numbers that can have fractional parts. Floats are commonly used for calculations involving decimals.

String:

A string is a sequence of characters enclosed within single ('') or double ("") quotation marks. Strings are used to represent textual data. They can contain letters, numbers, symbols, and spaces. Strings are versatile and are used for various purposes, such as storing names, messages, and more.

Note: Python is a dynamically typed language. Hence, we do not need to declare the type of variable during definition

Eg.

In [11]:
```
num = 1  #integer

dec = 5.4  #float

txt = 'Hello'  #string

print(f'Here {num} is an integer, {dec} is a float while {txt} s a string data typ

print(f'data type of num is: {type(num)}')
print(f'data type of dec is: {type(dec)}')
print(f'data type of txt is: {type(txt)}')
```

```
Here 1 is an integer, 5.4 is a float while Hello s a string data type
data type of num is: <class 'int'>
data type of dec is: <class 'float'>
data type of txt is: <class 'str'>
```

What is an expression made up of? What do all expressions do?

Ans.

An expression in programming is a combination of values, variables, operators, and function calls that can be evaluated to produce a result. Expressions represent computations and are used to perform operations on data.

An expression can be made up of the following components:

1. Values: These are constants like numbers or strings, which are the basic data elements in programming. For example, 5, "hello", and 3.14 are values.
2. Variables: These are named storage locations (identifiers) that hold values.
3. Operators: Operators are symbols or keywords that perform various operations on one or more values or variables. Examples of operators include + (addition), - (subtraction), * (multiplication), / (division), etc.
4. Function Calls: Functions are blocks of code that perform specific tasks.
   Functions can also be part of expressions, and their return values can be used as part of larger expressions.

All expressions in programming serve the common purpose of evaluating to a value. The value that an expression evaluates to can be of various types, such as numbers, strings, boolean values, objects, and more. The specific behavior of an expression depends on its components (values, variables, operators, and function calls) and how they are combine

Eg.

In [13]:
```
x = 5
y = 3
result = x + y * 2    #expression
print(result)         # Output: 11
```

```
11
```

5. This assignment statements, like spam = 10. What is the difference between an expression and a statement?

Ans.

Difference between Expression and Statement: The main difference between expressions and statements lies in their primary purpose and behavior:

Purpose:

Expressions are used to compute values, generate data, and produce results. Statements are used to perform actions, control program flow, and define the program's structure.

Value:

Expressions always have a value that they evaluate to. For example, 5 + 3 evaluates to 8. Statements may not necessarily have a value. An assignment statement like spam = 10 doesn't produce a value on its own; it assigns the value 10 to the variable spam.

Usage:

Expressions are often used within statements to provide values for calculations, comparisons, and decisions. Statements provide the structure and control for the program, defining what actions to take and in what order.

Eg.

In [18]:
```
spam = 10       # Assignment statement
new_spam = 3      # Assignment statement
result = spam + new_spam  # Expression: x + y
print(result)  # Output: 13

print('*********************************************')

if result > 10:  # Conditional statement
    print("Spams are greater than 10")
else:
    print("Spams under control")
```

```
13
*********************************************
Spams are greater than 10
```

6. After running the following code, what does the variable bacon contain?

bacon = 22

bacon + 1

Ans.

After running the given code, the variable bacon will still contain the value 22. The expression bacon + 1 is not assigned to any variable, so the result of the expression is calculated but not stored anywhere. If you want to update the value of bacon to include the result of the expression, you would need to explicitly assign the result back to the variable.

Execution in cell below

In [23]:
```
bacon = 22
print(f'the value of bacon is{bacon}')
print('*' * 50)
bacon + 1
print(f'the value of bacon is still {bacon}')
print('*' * 50)
bacon = bacon + 1  # Updating the value of bacon with the result of bacon + 1
print(f'the value of bacon after assignment is{bacon}')
```

```
the value of bacon is22
**************************************************
the value of bacon is still 22
**************************************************
the value of bacon after assignment is23
```

7. What should the values of the following two terms be?

'spam' + 'spamspam' 'spam' * 3

Ans.

The values of the two terms will be as follows:

1. `'spam' + 'spamspam'`: This expression performs string concatenation. It joins the string `'spam'` with the string `'spamspam'`, resulting in `'spamspamspam'`.
2. `'spam' * 3`: This expression performs string repetition. It repeats the string `'spam'` three times, resulting in `'spamspamspam'`.

So, the values of both terms will be `'spamspamspam'`.

Execution in cell below

In [26]:
```
# String concatenation
concatenated_string = 'spam' + 'spamspam'
print(concatenated_string)  # Output: 'spamspamspam'
print('*'*50)
# String repetition
repeated_string = 'spam' * 3
print(repeated_string)  # Output: 'spamspamspam'
```

```
spamspamspam
**************************************************
spamspamspam
```

8. Why is eggs a valid variable name while 100 is invalid?

Ans.

In Python, variable names must adhere to certain rules and conventions. The rules for naming variables include:

1. Start with a letter or underscore: Variable names must start with either a letter (a-z, A-Z) or an underscore (_).
2. Followed by letters, digits, or underscores: After the initial character, variable names can consist of letters, digits (0-9), or underscores.
3. Cannot start with a digit: Variable names cannot start with a digit (0-9).

Hence:

eggs: This is a valid variable name because it starts with a letter (e) and is followed by more letters. It adheres to the rules for variable naming in Python.

100: This is an invalid variable name because it starts with a digit (1). According to the rules, variable names cannot start with digits.

In [27]:
```
eggs = "This is a valid variable name."
print(eggs)
```

```
This is a valid variable name.
```

In [28]:
```
100 =  "This is an invalid variable name."
print(100)
```

```
  File "C:\Users\AMIT CHAURASIA\AppData\Local\Temp\ipykernel_20532\3584843282.py", line 1
    100 =  "This is an invalid variable name."
    ^
SyntaxError: cannot assign to literal
```

9. What three functions can be used to get the integer, floating-point number, or string version of a value?

Ans.

Integer Conversion: int()

The int() function is used to convert a value to an integer. It can convert a string containing an integer representation, a floating-point number, or another numeric type to an integer.

Floating-Point Conversion: float()

The float() function is used to convert a value to a floating-point number. It can convert a string containing a floating-point representation or another numeric type to a floating-point number.

String Conversion: str()

The str() function is used to convert a value to its string representation. It can convert any data type to a string.

Eg.

In [30]:
```
num_str = "42"
print(f'Initial type: {type(num_str)}')
str_int = int(num_str)  # Converts the string "42" to an integer 42
print(f'Changed type: {type(str_int)}')
print('*' * 50)

flt_str = "3.14"
print(f'Initial type: {type(flt_str)}')
str_float = float(flt_str)  # Converts the string "3.14" to a floating-point numbe
print(f'Changed type: {type(str_float)}')
print('*' * 50)

str_int = 42
print(f'Initial type: {type(str_int)}')
int_str = str(str_int)  # Converts the integer 42 to a string "42"
print(f'Change type: {type(int_str)}')
```

```
Initial type: <class 'str'>
Changed type: <class 'int'>
**************************************************
Initial type: <class 'str'>
Changed type: <class 'float'>
**************************************************
Initial type: <class 'int'>
Change type: <class 'str'>
```

10. Why does this expression cause an error? How can you fix it?

'I have eaten' + 99 + 'burritos.'

Ans.

Reason:

The expression 'I have eaten' + 99 + 'burritos.' causes an error because it attempts to concatenate a string ('I have eaten') with an integer (99) directly without any conversion. In Python, we cannot concatenate different data types like this without explicitly converting them to a compatible type first.

Fixing:

To fix this error, we need to ensure that all parts of the expression are of the same data type before performing the concatenation. We can convert the integer 99 to a string using the str() function or enclosing it between quotation marks (''), so that it can be concatenated with the other strings.

Eg.

In [33]:
```
Error = 'I have eaten' + 99 + 'burritos.'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20532\2041031896.py in <module>
----> 1 Error = 'I have eaten' + 99 + 'burritos.'

TypeError: can only concatenate str (not "int") to str
```

In [32]:

```
result = 'I have eaten ' + str(99) + ' burritos.'
print(result)

print('Or '+ "*"*50)

result2 = 'I have eaten ' + '99' + ' burritos.'
print(result2)
```

```
I have eaten 99 burritos.
Or **************************************************
I have eaten 99 burritos.
```